



Алгоритмы и структуры данных

Лекция 8. Связанные списки (2)

Антон Штанюк (к.т.н, доцент)

15 апреля 2022 г.

Нижегородский государственный технический университет им. Р.Е. Алексеева
Институт радиоэлектроники информационных технологий
Кафедра "Компьютерные технологии в проектировании и производстве"

Двусвязный список

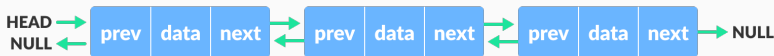
Программная реализация

Список литературы

Двусвязный список

Особенности двусвязных списков

Для двусвязных списков характерно использование двух указателей в каждом элементе: на следующий элемент и на предыдущий. Это позволяет двигаться по списку в двух направлениях: от головы к хвосту и от хвоста к голове.



Программная реализация

Берем за основу реализацию односвязного списка и расширяем её. В структуру узла **ITEM** необходимо добавить указатель **prev** на предыдущий элемент.

В класс добавим указатель **current**, который позволит реализовывать **итератор**.

Расширим интерфейс класса, добавив возможность печатать элементы списка в реверсивном порядке, создавать копии списка, вставлять элементы в произвольную позицию.

```
template<typename T>
class TDList {
private:
    struct ITEM {
        T data;
        ITEM * next;
        ITEM * prev;
    };
    TDList::ITEM* create(const T&);
    ITEM *head;
    ITEM *tail;
    ITEM *current;
    ...
};
```

```
...  
    TDList():head(nullptr),tail(nullptr),current(nullptr){}  
    TDList(const T&);  
    TDList(const TDList&);  
    ~TDList();  
    void addTail(const T&);  
    void addHead(const T&);  
    T rmHead();  
    T rmTail();  
    T getNext();  
    void reset();  
    void print() const;  
    void printReverse() const;  
    bool isEmpty() const;  
    void insert(const T&, const T&);
```


Здесь отличий от односвязного списка нет.

```
template<typename T>
TDList<T>::TDList(const T& data) {
    head=tail=create(data);
    current=nullptr;
}
template<typename T>
TDList<T>::~~TDList {
    while(head)
        rmHead();
}
```

```
template<typename T>
typename TDList<T>::ITEM* TDList<T>::create(const T& data) {
    ITEM *item=new ITEM;
    item->data=data;
    item->next=nullptr;
    item->prev=nullptr;
    return item;
}
```

```
template<typename T>
void TDList<T>::addTail(const T& data) {
    if(tail && head) {
        tail->next=create(data);
        tail->next->prev=tail;
        tail=tail->next;
    }
    else {
        head=create(data);
        tail=head;
    }
}
```

```
template<typename T>
void TDList<T>::addHead(const T& data) {
    if(tail && head) {
        ITEM *temp=create(data);
        temp->next=head;
        temp->next->prev=temp;
        head=temp;
    }
    else {
        head=create(data);
        tail=head;
    }
}
```

```
template<typename T>
T TDList<T>::rmHead() {
    if(head) {
        ITEM *temp=head->next;
        if(temp)
            temp->prev=nullptr;
        T data=head->data;
        delete head;
        head=temp;
        return data;
    }
    else {
        throw std::string("Empty!");
    }
}
```

```
template<typename T>
T TDList<T>::rmTail() {
    if(head && tail) {
        ITEM *temp=tail->prev;
        if(temp)
            temp->next=nullptr;
        T data=tail->data;
        delete tail;
        tail=temp;
        return data;
    }
    else {
        throw std::string("Empty!");
    }
}
```

```
template<typename T>
bool TDList<T>::isEmpty() const {
    return !head;
}
```

Метод печати от начала списка до конца не отличается от аналогичного, реализованного в односвязном списке

```
template<typename T>
void TDList<T>::print() const {
    ITEM *temp=head;
    while(temp) {
        std::cout<<temp->data<<" ";
        temp=temp->next;
    }
    std::cout<<std::endl;
}
```



```
template<typename T>
void TDList<T>::printReverse() const {
    ITEM *temp=tail;
    while(temp) {
        std::cout<<temp->data<<" ";
        temp=temp->prev;
    }
    std::cout<<std::endl;
}
```

Итератор - это "умный" указатель, позволяющий перебирать элементы в определенном направлении. Мы храним адрес текущего элемента в **current** и сдвигаем его на следующий элемент при вызове **getNext**

```
template<typename T>
T TDList<T>::getNext() {
    if(head && tail) {
        if(!current)
            current=head;
        else if(current->next)
            current=current->next;
        else
            throw std::string("The_end!");
        return current->data;
    }
    else {
        throw std::string("Empty!");
    }
}
```

Используя итератор, можно реализовать конструктор копирования списка

```
template<typename T>
TDList<T>::TDList(const TDList& list) {
    try {
        const_cast<TDList&>(list).reset();
        while(true) {
            T val = const_cast<TDList&>(list).getNext();
            addTail(val);
        }
    }
    catch(std::string& err) {
    }
    current=nullptr;
}
```

Специальный метод позволяет сбросить итератор, то есть возвращает его в начало списка

```
template<typename T>
void TDList<T>::reset() {
    current=nullptr;
}
```

Реализуем метод вставки в произвольную позицию, **перед** заданным значением элемента. В списке ищется элемент со значением **data** и перед ним осуществляется вставка элемента с **value**. Если элемент не найдет, вставка происходит в хвост списка.

```
template<typename T>
void TDList<T>::insert(const T& data, const T& value) {
    ITEM * temp = head;
    ITEM * item = create(value);
    while(temp && temp->data != data)
        temp=temp->next;

    if(!temp && head) { // вставка в конец
        tail->next = item;
        tail->next->prev=tail;
        tail=item;
    }
    ...
}
```

Метод вставки в произвольную позицию списка

```
...
else if(!temp && !head) { // вставка в пустой список
    head=tail=item;
}
else if(!temp->prev) { // вставка в начало
    temp->prev=item;
    item->next=temp;
    head=item;
}
else { // вставка в середину
    temp->prev->next=item;
    item->prev=temp->prev;
    item->next=temp;
    temp->prev=item;
}
}
```

```
#include <iostream>
#include "tdlist.h"

int main() {
    int i;
    TDList<int> list;
    for(i=1;i<10;i++) {
        list.addTail(i);
        list.print();
    }
    for(i=10;i<15;i++) {
        list.addHead(i);
        list.print();
    }

    ...
}
```

```
...
try {
    while(list.rmHead())
        list.print();
}
catch(std::string& err) {
    std::cout << err;
}

for(i=1;i<10;i++) {
    list.addTail(i);
}
list.printReverse();
...
```



```
...
    try {
        while(true) {
            int val = list.getNext();
            std::cout << val << ", ";
        }
    }
    catch(std::string& err) {
        std::cout << err;
    }

    list.reset();
    std::cout << std::endl;

    TDList<int> list2(list);
    list2.print();

    list2.rmTail();
    list2.print();
    ...
```

Пример использования двусвязного списка

```
...
TDList<int> list3;
list3.insert(5,5);
list3.print();
list3.insert(5,4);
list3.print();
list3.insert(5,3);
list3.print();
list3.insert(6,6);
list3.print();
list3.insert(5,1);
list3.print();

return 0;
}
```

Пример использования двусвязного списка

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
10 1 2 3 4 5 6 7 8 9
11 10 1 2 3 4 5 6 7 8 9
12 11 10 1 2 3 4 5 6 7 8 9
13 12 11 10 1 2 3 4 5 6 7 8 9
14 13 12 11 10 1 2 3 4 5 6 7 8 9
```

Пример использования двусвязного списка

13 12 11 10 1 2 3 4 5 6 7 8 9

12 11 10 1 2 3 4 5 6 7 8 9

11 10 1 2 3 4 5 6 7 8 9

10 1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

2 3 4 5 6 7 8 9

3 4 5 6 7 8 9

4 5 6 7 8 9

5 6 7 8 9

6 7 8 9

7 8 9

8 9

9

Empty!9 8 7 6 5 4 3 2 1

1, 2, 3, 4, 5, 6, 7, 8, 9, The end!

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8

5






4 5






4 3 5

4 3 5 6

4 3 1 5 6

Список литературы

-  Кормен Т.,Лейзерсон Ч., Ривест Р.
Алгоритмы: построение и анализ
МЦНМО, Москва, 2000
-  Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы:
построение и анализ.
2-е изд. — М.: «Вильямс», 2006
-  Википедия
Алгоритм
<http://ru.wikipedia.org/wiki/Алгоритм>
-  Википедия
Список алгоритмов
http://ru.wikipedia.org/wiki/Список_алгоритмов
-  Традиция
Задача коммивояжёра
<http://traditio.ru/wiki/Задача>

-  Википедия
NP-полная задача
<http://ru.wikipedia.org/wiki/NP-полная>
-  Серджвик Р.
Фундаментальные алгоритмы на C++. Части 1-4
Diasoft, 2001
-  Седжвик Р.
Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск
СПб.: ДиаСофтЮП, 2003
-  Седжвик Р.
Фундаментальные алгоритмы на C. Алгоритмы на графах
СПб.: ДиаСофтЮП, 2003
-  Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.
Издательский дом «Вильямс», 2000



Кнут Д.

Искусство программирования, том 1. Основные алгоритмы

3-е изд. — М.: «Вильямс», 2006



Кнут Д.

Искусство программирования, том 2. Получисленные методы

3-е изд. — М.: «Вильямс», 2007



Кнут Д.

Искусство программирования, том 3. Сортировка и поиск

2-е изд. — М.: «Вильямс», 2007



Кнут Д.

Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений

М.: «Вильямс», 2007



Кнут Д.

Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации

М.: «Вильямс», 2007